SmartSDLC-AI-Enhanced Software Development Lifecycle

Project Description:

SmartSDLC uses the Granite model from Hugging Face to speed up software development. It lets users upload PDFs, generate clear requirements, turn prompts into code, create tests, fix bugs, write docs, and chat with an AI helper. This project will be deployed in Google Colab using Granite for easy setup and reliable performance.

My team has successfully enrolled for the project. Find the team detailsbelow.

Team ID: NM2025TMID06245

Team Size: 4

Team Leader: RISHABAVARTHINI V

Team member: GOWRI S

Team member: MANOJINI A

Team member: SOWNDARYA S

Activity-1: Exploring Naan Mudhalavan Smart Interz Portal.

* + Search for “Naan Mudhalavan Smart Interz” Portal in any Browser.
  + Then Click on the first link.
  + ([NaanmudhalvanSmartinternz](https://naanmudhalvan.smartinternz.com/)) Then login with your details.
  + Then you will be redirected to your account then click on “Projects”
  + Section.
  + There you can see which project you have enrolled in here it is “SmartSDLC-AI-Enhanced Software Development Lifecycle”.
  + Then click on “Access Resources” and go to the “Guided Project” Section.
  + Click on the “Go to workspace” section.
  + Then you can find the detailed explanation of Generative AI Project using IBM WatsonX API key.
  + Click on “Project Workspace”, there you can find your project progress and Place to upload “Demo link”.
  + Now we have gone through portal understanding, now lets find a IBM granite model fromhugging face to integrate in our project.

Activity-2: Choose a IBM Granite model From Hugging Face.

Search for “Hugging face” in any browser.

Then click on the first link ([HuggingFace](https://huggingface.co/)), then click on signup and create your own account in Hugging Face.

Then search for “IBM-Granite models” and choose any model.

Here for this project we are using “granite-3.2-2b-instruct” which is compatible fast and light weight.

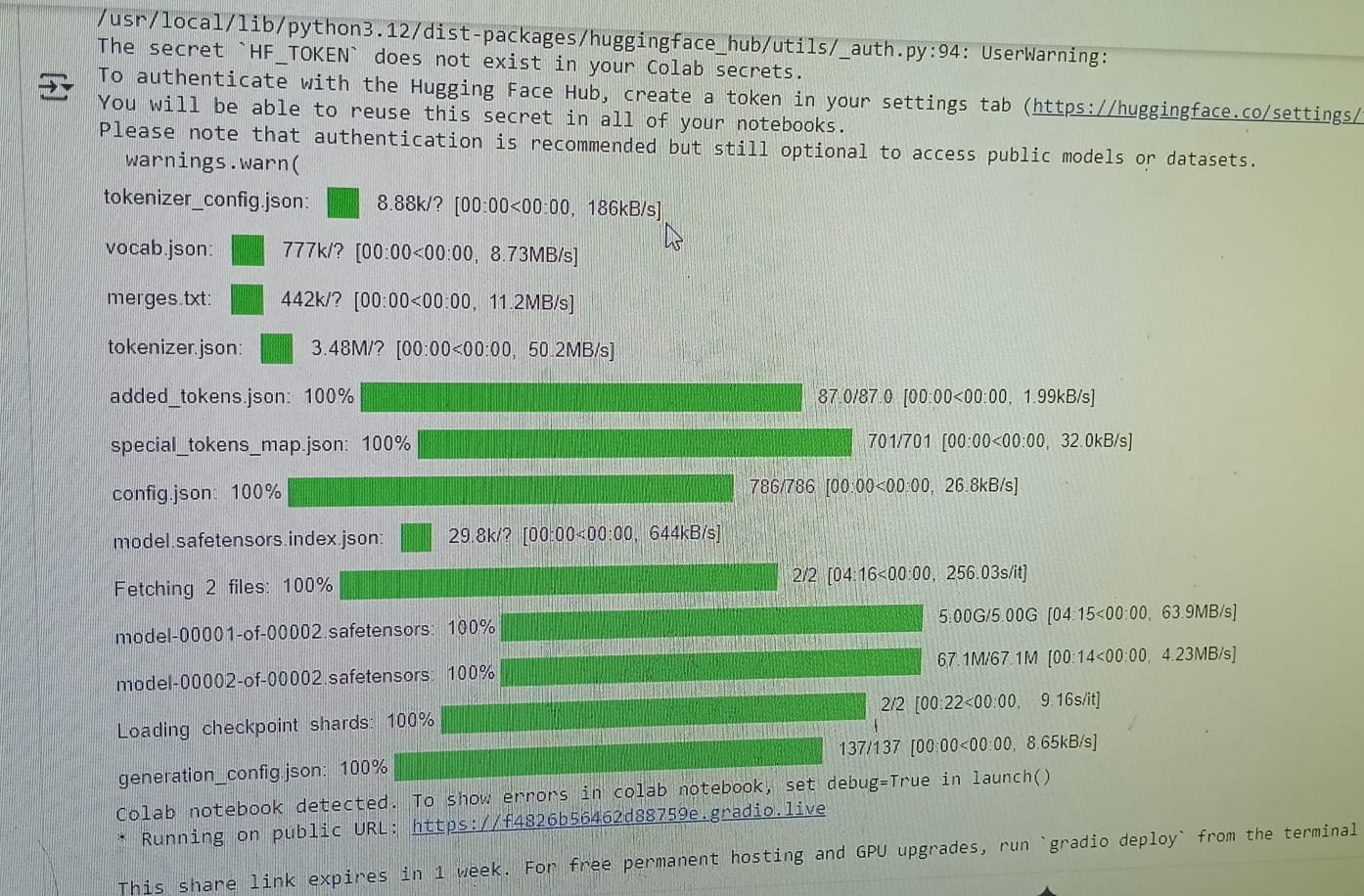
Now we will start building our project in Google collab.

Activity-3: Running Application in Google Collab.

* Search for “Google collab” in any browser.
* Click on the first link ([GoogleColab](https://colab.research.google.com/)), then click on “Files” and then “Open Notebook”.
* Click on “New Notebook”.
* Change the title of the notebook “Untitled” to “Health AI”. Then click on “Runtime”, then go to “Change Runtime Type”.
* Choose “T4 GPU” and click on “Save”.
* Run this command in first cell“!pipinstall transformers torch gradio PyPDF2 -q”.
* To install the required libraries to run our application.
* Then run the rest of the code in the next cell
* #import Libraries
* import gradio as gr
* import torch
* from transformers import AutoTokenizer, AutoModelForCausalLM
* import PyPDF2
* #load model and tokenizer
* model\_name = "ibm-granite/granite-3.2-2b-instruct"
* tokenizer = AutoTokenizer.from\_pretrained(model\_name)
* model = AutoModelForCausalLM.from\_pretrained(
* model\_name,
* torch\_dtype=torch.float16 if torch.cuda.is\_available() else torch.float32,
* device\_map="auto" if torch.cuda.is\_available() else None
* )
* if tokenizer.pad\_token is None:
* tokenizer.pad\_token = tokenizer.eos\_token
* def generate\_response(prompt, max\_length=1024):
* inputs = tokenizer(prompt, return\_tensors="pt", truncation=True, max\_length=512)
* if torch.cuda.is\_available():
* inputs = {k: v.to(model.device) for k, v in inputs.items()}
* with torch.no\_grad():
* outputs = model.generate(
* \*\*inputs,
* max\_length=max\_length,
* temperature=0.7,
* do\_sample=True,
* pad\_token\_id=tokenizer.eos\_token\_id
* )
* response = tokenizer.decode(outputs[0], skip\_special\_tokens=True)
* response = response.replace(prompt, "").strip()
* return response
* def extract\_text\_from\_pdf(pdf\_file):
* if pdf\_file is None:
* return ""
* try:
* pdf\_reader = PyPDF2.PdfReader(pdf\_file)
* text = ""
* for page in pdf\_reader.pages:
* text += page.extract\_text() + "\n"
* return text
* except Exception as e:
* return f"Error Reading Pdf : {str(e)}"
* def requirement\_analysis(pdf\_file, prompt\_text):
* if pdf\_file is not None:
* content = extract\_text\_from\_pdf(pdf\_file)
* analysis\_prompt = f"""
* Analyze the following document and extract key software requirements.
* Organize them into:
* - Functional Requirements
* - Non-Functional Requirements
* - Technical Specifications
* Document Content:
* {content}
* """
* else:
* analysis\_prompt = f"""
* Analyze the following requirements and organize them into:
* - Functional Requirements
* - Non-Functional Requirements
* - Technical Specifications
* Requirements:
* {prompt\_text}
* """
* return generate\_response(analysis\_prompt, max\_length=1200)
* def code\_generation(prompt, language):
* code\_prompt = f"""
* Based on the following requirement, generate {language} code:
* Requirement:
* {prompt}
* Code:
* """
* return generate\_response(code\_prompt, max\_length=1200)
* #Create Gradio interface
* with gr.Blocks() as app:
* gr.Markdown("# 🧠 Smart SDLC Assistant - Code Analysis & Generator")
* with gr.Tabs():
* with gr.TabItem("Requirement Analysis"):
* with gr.Row():
* with gr.Column():
* pdf\_upload = gr.File(label="Upload PDF", file\_types=[".pdf"])
* keywords\_input = gr.Textbox(label="Or write requirement here", placeholder="Describe your software requirements...", lines=3)
* analysis\_tips\_btn = gr.Button("Analyze")
* with gr.Column():
* tips\_output = gr.Textbox(label="Requirements Analysis", lines=20)
* analysis\_tips\_btn.click(requirement\_analysis, inputs=[pdf\_upload, keywords\_input], outputs=tips\_output)
* with gr.TabItem("Code Generation"):
* with gr.Row():
* with gr.Column():
* code\_prompt = gr.Textbox(label="Code Requirements", placeholder="Describe what code you want to generate...", lines=5)
* language\_dropdown = gr.Dropdown(
* choices=["Python","JavaScript","Java","C++","C#","PHP","Go","Rust"],
* label="Programming Language",
* value="Python"
* )
* generate\_btn = gr.Button("Generate Code")
* with gr.Column():
* code\_output = gr.Textbox(label="Generated Code", lines=20)
* generate\_btn.click(code\_generation, inputs=[code\_prompt, language\_dropdown], outputs=code\_output)
* app.launch(share=True)
* You can find the code here in this link: SmartSDLC

Output:

* Now you can see our model is being Downloaded and application is running.



* Click on the URl to open the Gradio Application [https://f4826b56462d88759e.gradio.live](https://f4826b56462d88759e.gradio.live/)
* click on the link.

Activity-4: Upload Your Project in GitHub.

* Search for “GitHub” in any browser, then click on the first link ([GitHub](https://github.com/)).
* Then click on “Signup” and create your own account in GitHub.
* If you already have an account click on “Sign in”.
* Click on “Create repository”.

In “General” Name your repo. (Here I have given” <https://github.com/23ugbca35dkgac-design/SmartSDLC_AI.git>”

as my repo name and it is available)

* In “Configurations” Turn On “Add readme” file Option.
* Now Download your code from Google collab by Clicking on “File”, then Goto “Download” then download as “.py”.
* Then your repository is created, then Click on “Add file” Option.
* Then Click “Upload files” to upload your files.
* Click on “choose your file”.
* Choose your project file and click on “Open”.
* After your file has Uploaded Click on “Commit changes”.
* THANKS TO SMART BRIDGE TEAM FOR GIVEN THIS OPPORTUNITY BY
* NM2025TMID06245– PROJECT
* TEAM